**12/13/2024**

**CIS450/ECE478 Final Project**

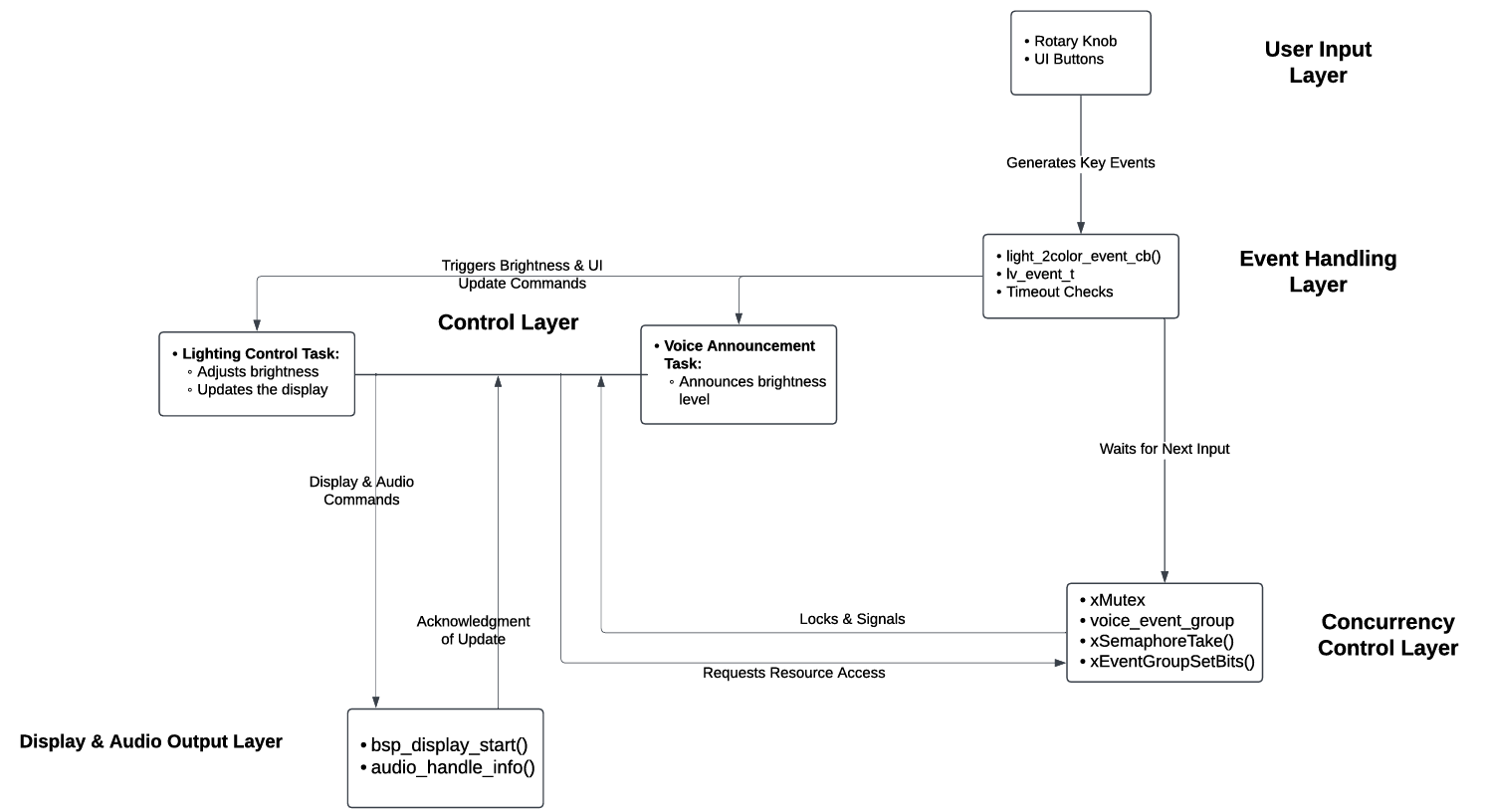
**ESP32 Light Multithreading**

**OS team/Team 11**

**Hamzah Ismail, Hawraa Ismail, Christopher Woods**

**Dr. Roy Probir**

**1.System Architecture:**

 Figure 1.1: System’s task structure and interaction

**User Input Layer**

The User Input Layer consists of a rotary knob and UI buttons. The rotary knob provides the user with the ability to change the brightness of the LEDs in 25% steps, while UI buttons provide the means for interaction with the system, such as switching between warm and cool light modes or going back to the main menu. These user inputs generate events that are processed in the following layers.

**Event Handling Layer**

In the Event Handling Layer, the light\_2color\_event\_cb() function captures user input events, such as knob rotation or button press, and processes them according to the logic of the system. If the knob is rotated right and brightness is less than 100%, it increases the brightness by 25%. On the other hand, if the knob is rotated left, it decreases by 25% if the brightness is greater than 0%. For every brightness adjustment, the function announce\_brightness() is called, which is a signal for the system to trigger voice feedback. Timeout checks, such as is\_time\_out(&time\_500ms), are employed to debounce input and avoid repeated triggers.

**Control Layer**

The Control Layer is made up of the two main tasks: the Lighting Control and Voice Announcement. This task of Lighting Control will modify the light\_set\_conf.light\_pwm variable with a new brightness. Further, it updates the display using functions in LittlevGL such as lv\_label\_set\_text\_fmt and updates the LED lighting through bsp\_led\_rgb\_set(). It maps a brightness value to RGB LED intensities. The Voice Announcement task runs independently, signaled by the announce\_brightness() function setting a global flag in an Event Group. This flag is detected by the Voice Announcement task, using xEventGroupWaitBits, which then locks the shared current\_brightness variable using xMutex for thread safety, retrieves the value of the current brightness, and plays the corresponding audio file using audio\_handle\_info().

**Concurrency Control Layer**

The Concurrency Control Layer provides smooth multitasking and safe use of shared resources. In this layer, the xMutex semaphore is implemented to avoid race conditions when several tasks access current\_brightness. An Event Group provides a way for the different tasks to communicate with one another: voice\_event\_group. In addition, the xEventGroupWaitBits function of Event Group allows the Voice Announcement task to wait on a signal provided by announce\_brightness() for synchronization between these tasks with no busy waiting. With the aid of all these primitives for synchronization, smooth operation of the system without the effects of conflict of tasks becomes feasible.

**Display & Audio Output Layer**

The Display & Audio Output Layer: This layer manages the system's visual and audio outputs. Major updates regarding the brightness level and the on/off state of the LEDs are changed using the functions lv\_img\_set\_src() and lv\_label\_set\_text\_fmt(). This provides auditory feedback through audio\_handle\_info() to produce an audio file that represents the current brightness. This provides a feedback loop that adds to the system's usability by including both visual and audio cues.

**System Flow Case**

Here is a step-by-step example of how the system behaves when the user interacts with it:

**Case: User Adjusts Brightness Using the Knob**

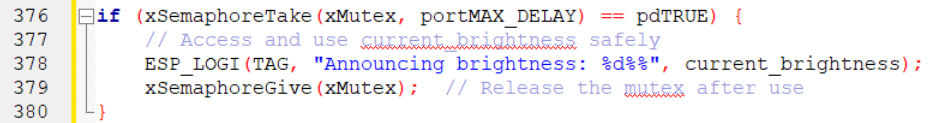
1. **User Input Layer:**
   * The rotary knob is turned to the right.
2. **Event Handling Layer:**
   * light\_2color\_event\_cb() captures the knob rotation event.
   * The timeout check ensures the event is not a duplicate.
   * Brightness is increased by 25% (e.g., from 50% to 75%).
   * announce\_brightness(75) is called.
3. **Control Layer (Lighting Control Task):**
   * The light\_set\_conf.light\_pwm variable is updated to 75%.
   * The display updates to reflect the new brightness level (e.g., the brightness label changes to "75%").
   * The RGB LEDs are updated using bsp\_led\_rgb\_set().
4. **Concurrency Control Layer:**
   * announce\_brightness() sets the VOICE\_EVENT\_BIT in voice\_event\_group.
   * The shared current\_brightness variable is accessed using xMutex for thread safety.
5. **Control Layer (Voice Announcement Task):**
   * The voice announcement task is unblocked when VOICE\_EVENT\_BIT is set.
   * It announces "75%" using audio\_handle\_info().
6. **Display & Audio Output Layer:**
   * The display remains updated with the new brightness level.
   * Audio feedback is played to announce the brightness level.
7. **Feedback Loop:**
   * The system waits for further input, such as another knob rotation or button press.

**2.Concurrency Control Explanation:**

The system uses mutexes and semaphores to manage access to shared resources and ensure safe task execution in a multitasking environment. Here is a detailed breakdown of their roles:

**1. Mutex (xMutex):**

* The mutex xMutex protects the shared resource, current\_brightness. Both the Lighting Control Task and the Voice Announcement Task will be interacting with this variable; hence, the mutex ensures that only one task can access and modify it at a time.
* **Usage in Code:**
  + When a brightness adjustment is made, the announce\_brightness() function uses xMutex to safely set the current\_brightness variable. The mutex makes sure that no other task (like the Voice Announcement Task) modifies the variable simultaneously.
  + In the Voice Announcement Task, xSemaphoreTake() is used to acquire the mutex before reading the current\_brightness value. Once the task is finished using the variable, it releases the mutex with xSemaphoreGive().
* **Purpose:**
  + Prevents race conditions where multiple tasks attempt to read or write to current\_brightness at the same time.
  + Guarantees data consistency by allowing only one task to access the resource at any given moment.
* **Example Usage:**
  + In the voice\_announcement\_task(), the mutex is used to safely access current\_brightness:



**Event Groups (voice\_event\_group):**

The event group (voice\_event\_group) is used for signaling between tasks. It ensures proper synchronization between the Lighting Control Task and the Voice Announcement Task. Specifically, it uses an event bit (VOICE\_EVENT\_BIT) to notify the Voice Announcement Task whenever a brightness change occurs.

* **Usage in Code:**
  + In the announce\_brightness() function, when the brightness is updated, the VOICE\_EVENT\_BIT in the event group is set to signal the Voice Announcement Task.
  + The Voice Announcement Task waits for this event bit using xEventGroupWaitBits(). This allows the task to remain idle until a brightness change occurs, improving efficiency by avoiding unnecessary CPU usage.
  + After processing the event, the event bit is automatically cleared to prepare for the next signal.
* **Purpose:**
  + Allows efficient task synchronization by notifying the Voice Announcement Task only when a brightness update occurs.
  + Prevents busy waiting, as the Voice Announcement Task remains blocked until the event bit is set.
  + Ensures proper task coordination by maintaining a strict sequence: brightness update → signal → voice announcement.

**3.User Guide:**

The lighting is controlled on a built-in, easy to use panel: The knob should be rotated clockwise to increase the light by 25% and counterclockwise to decrease. The screen will show the updated light level, such as "75%," and announce it like "Seventy-five percent" by playing a specified audio file.

Press the button to switch between warm and cool light modes. The screen will reflect visually on the currently activated mode. Press and hold the button, long press to go back to the main menu. The display clears the current settings and goes back to the main interface.

Voice announcements will be made for changes in brightness, and therefore the system can be used completely without relying on visual cues. Ensure smooth operation by allowing brief pauses between knob rotations or button presses.

**4.Bonus Feature Documentation:**

Description, design considerations, and usage instructions for the additional feature (if implemented).

The idea is to multi-thread the thermostat controls with the brightness of the light, allowing the brightness and thermostat UI to be updated simultaneously.  
  
The attempt was unsuccessful, but the idea is documented below:

 //Creates RTOS event group

static EventGroupHandle\_t control\_event\_group;

//create tasks for temperature control and for brightness

xTaskCreate(temperature\_control\_task, "TempTask", 2048, NULL, 5, NULL); xTaskCreate(brightness\_control\_task, "BrightnessTask", 2048, NULL, 5, NULL);

void brightness\_control\_task(void \*param) {     while (true) {         xEventGroupWaitBits(control\_event\_group, BRIGHTNESS\_EVENT\_BIT, pdTRUE, pdFALSE, portMAX\_DELAY);

ESP\_LOGI(TAG, "Brightness changed: %d%%", current\_brightness);

if (key == LV\_KEY\_RIGHT && light\_set\_conf.light\_pwm < 100)

{     light\_set\_conf.light\_pwm += 25; }

else if (key == LV\_KEY\_LEFT && light\_set\_conf.light\_pwm > 0)

{     light\_set\_conf.light\_pwm -= 25; }

lv\_arc\_set\_value(temp\_arc, light\_set\_conf.light\_pwm);

lv\_roller\_set\_selected(temp\_wheel, (light\_set\_conf.light\_pwm - 19), LV\_ANIM\_ON);

} }

void temperature\_control\_task(void \*param) {

while (true) {

  xEventGroupWaitBits(control\_event\_group, TEMP\_EVENT\_BIT, pdTRUE, pdFALSE, portMAX\_DELAY);

ESP\_LOGI(TAG, "Temperature changed: %d°C", current\_temperature);

if (current\_temperature < 20)

{ light\_set\_conf.light\_pwm = 25; }

else if (current\_temperature < 25)

{ light\_set\_conf.light\_pwm = 50; }

else if (current\_temperature < 30)

{ light\_set\_conf.light\_pwm = 75; }

else { light\_set\_conf.light\_pwm = 100; }

event xEventGroupSetBits(control\_event\_group, BRIGHTNESS\_EVENT\_BIT);

} }

xEventGroupSetBits(control\_event\_group, TEMP\_EVENT\_BIT);